DICOM in Dart (DCMiD)
Group 13 Final Project Report

Damish Shah, Danielle Tinio

Mentor: James Philbin PhD

May 2014

Abstract. Medical imaging data has become crucial to the proper function of healthcare systems. It is crucial that information systems be created that allow end users to access and edit DICOM data in a fast, secure, and intuitive manner. A novel application framework was developed to parse binary DICOM medical image data and display it in a collapsible tree structure in a web browser. The application can parse DICOM metadata in under 3 seconds for a variety of file sizes and in under 1 second for small files (under 1 megabyte). A system architecture was written that includes a base class for handling the binary data, parsers that create human readable objects out of the data, as well as a server and client for user access of DICOM imaging studies.

1 Introduction

Digital Imaging and Communication in Medicine (DICOM) [1] is the standard for storing and transmitting medical imaging studies such as CT and MRI scans. Nearly all healthcare systems use the DICOM standard in their medical imaging equipment and information systems. It is crucial that information systems be created that allow end users to access and edit DICOM data in a fast, secure, and intuitive manner.

1.1 The DICOM Standard

Learning about the structure of the DICOM Standard and its nuances was the most crucial part in writing the web application. The DICOM Standard specifies the information entity hierarchy that a DICOM file should be stored in. At the very top it specifies a Patient which has a list of Studies. The hierarchy continues through Series followed by Instance until image frames occur at the lowest level, each stored within a separate Instance. At each level there exist information attributes about the current level, as well as a list of objects at the next lowest level of the hierarchy.

The DICOM Standard also specifies the way in which each of the information entities need to be stored in memory. A binary DICOM file has a top level Dataset that contains a list of Attributes. It is important to recognize the structure of an Attribute as the lowest level of storage specification in the DICOM Standard. Every bit of information in a DICOM file is stored as an Attribute. An Attribute is structured as shown in figure 1. An Attribute contains a unique, 4 byte, identifying Tag depending on the data that is being stored in the Attribute. The Tag contains a group part and an element part each of which is 2 bytes long. An Attribute also contains a 2 or 4 byte Value Representation that specifies the encoding of the value field contents and which parser should be used to read this Attribute. Next in the Attribute is a length field that contains the length in bytes of the Value Field. Finally, the Attribute has a Value Field that stores the actual data of the Attribute.

A Value Field within an Attribute can also contain a Sequence, which is the way that an element in the top level information entity hierarchy is stored. A Study, Series, and Instance is stored as a Sequence in memory. A Sequence is defined as a list of Items and each Item contains a Dataset allowing the entire data structure to be recursive. The Dataset structure and relationships are shown in figure 2.

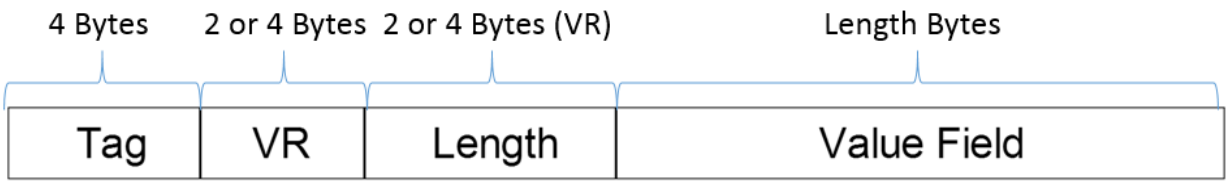


Figure 1: Structure of a DICOM Attribute.

Contains an identifying Tag, a Value Representation that defines the encoding of the Value Field, Length of the Value Field, and a Value Field that contains the actual information stored in the attribute.

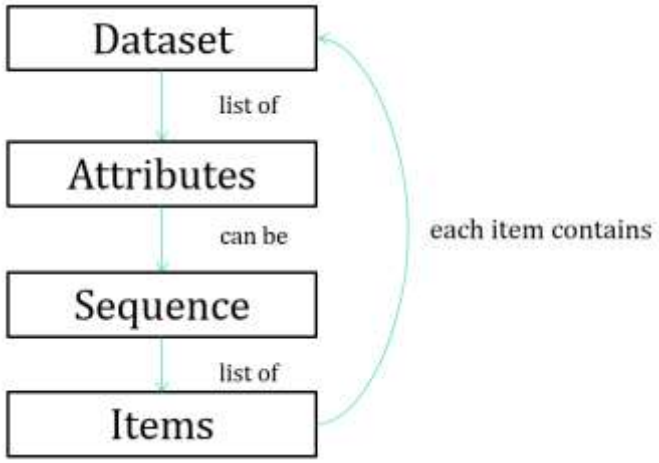


Figure 2: Data structure of DICOM data.

From the image, we can see that the relationship is recursive. A dataset has a list of attributes which can be a sequence. Sequences have a list of items, which each contain a dataset.

2 Problem

While software for viewing DICOM studies exists, the reading and writing of binary DICOM data can be improved.

2.1 Create a Zero-Footprint Web Application

Dart, Google's new, class-based, single-inheritance, object-oriented programming language designed explicitly for creating web applications is proposed to handle this task. One goal is to demonstrate the viability of retrieving and displaying binary DICOM in a web application. Using the nested structure shown in Figure 2, the client will display DICOM data in a nested tree-table format, allowing the user to expand and collapse the information levels.

A zero-footprint web client offers many advantages over traditional software applications. It allows the application to stream data and prevents the need for sensitive, HIPAA covered patient data to be stored on the end point device. It also allows the web application to be maintained and upgraded without the need for access control on the end point device, lowering management costs. There is also an opportunity for improved security since the data can be encrypted before transmission.

2.2 Decrease Development Overhead

One of the goals for the open-source DICOM web viewer was to decrease the amount of time and effort required for developers to contribute to and continue the project. The Dart programming language is a fast and intuitive tool to create and maintain the web application. Dart is similar to many object-oriented programming languages, such as Java, C++, and C#. In contrast to JavaScript, the primary language used to create web applications, Dart is easy to learn and use because it is class-based and has good library and package management functionality. Like JavaScript, it is dynamically typed and functions and their closures are first class.

2.3 Increase Parsing Speed

Increased speed of DICOM data management is significant for users such as large hospitals, which process hundreds of images daily, and regional Health Information Exchanges, that manage millions of images. As the use of medical imaging studies increases, it becomes more important to access and manipulate this data faster. Dart has an advantage over JavaScript in that it performs better on several major benchmarks.[2] Native Dart is currently only supported in Google based browsers, but is also able to compile to JavaScript to run in other web browsers. Even when compiled to JavaScript through the dart2js compiler, Dart still maintains a large speed advantage over native JavaScript. The goal was to be able to parse all of the DICOM test files in under 3 seconds.

3 Methods

3.1 Dart Programming Language

The main goal was to write a general open source application and development framework for working with binary DICOM files. A base class for handling binary DICOM data was written called ByteBuf. A ByteBuf object is created from a file and is not DICOM specific. ByteBuf contains binary parsers and handling tools that will work on any type of file. For DICOM specific applications a Dataset class was written that contains DICOM specific parsers and formatted readers for attributes. This is also where the recursive logic for reading sequences is defined. A Dataset can be used to read a single DICOM file stored in a ByteBuf object in its entirety. At its essence a DICOM file can be parsed with the following code, where bytes is the content of the file read as bytes.

```
ByteBuf bytebuffer = new ByteBuf.from(bytes);
Dataset dataset = new Dataset.Fmi(bytebuffer);
dataset.read();
```

A Dataset contains lists of tags, value representations, and value fields, to simulate a list of attributes. Once the read() method is called on a Dataset created from a ByteBuf, the Dataset recursively reads through the contents of its ByteBuf to populate those lists. The parsed data is then handled through the built in lists and their methods.

In an attempt to write general code many helper classes have been written using Dart along with the main classes that make working with DICOM and other binary file types much easier for developers. As an example classes were written that help in producing and dealing with errors, classes for many of the custom DICOM elements such as BulkDataReference and DicomDateTime, and code that defines the top-level information entity hierarchy elements. These tools will likely help future developers on this project in the continuation of this code into a fully featured web application.

3.2 Polymer Library

The polymer library provides the tools to create elements that can be reusable in web applications. Instead of looking at applications as a block of script, it demonstrates that applications can be written as a combination of element building blocks that are coded through HTML and Dart scripts. Polymer provides a framework for two-way data binding between html and Dart. It allows us to treat HTML templates like Dart classes and communicate with them through shared variables. In particular, the polymer element, table-row, is used to add and interact with children elements. When the program encounters a sequence it adds the sequence as a child to the polymer element at the current level of the recursion. The polymer element has methods for handling children addition and also handling click events. When the element title is clicked it toggles the hidden property of all of its children, which gives us the collapsible table functionality.

3.3 Server and Client Programming

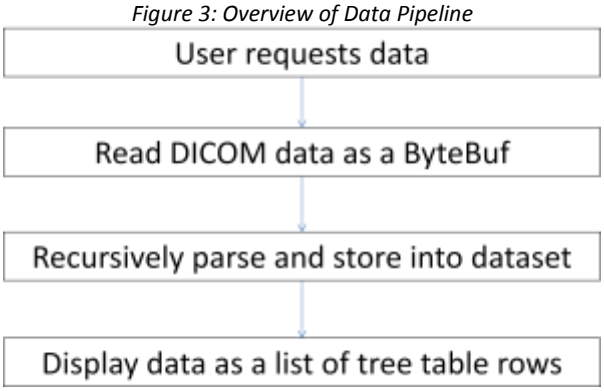
One of the benefits of using Dart to create the web application is that both server and client side code can be written in dart. The server handles requests made from the client code for specific DICOM files and returns the appropriate bytebuffer containing the byte data from the file. Be careful not to confuse this built in Dart class of bytebuffer for handling Dart bytedata with the ByteBuf class that was written for reading data stored as bytes in this project. The best practices for coding server-client relationships in Dart is still not fully understood since the programming language is so young. The server returns the bytebuffer containing the DICOM file stored as a bytedata object after the request is made by the client. Then on the client side the data is stored as a ByteBuf and then subsequently turned into a Dataset which is then parsed. This parsed data is finally displayed on the client as a nested set of table-row polymer elements.

3.4 Data Pipeline

Figure 3 demonstrates the pipeline of how an end-point user is able to see the DICOM data. This is a result of building binary parsers, string parsers, the main and helper classes, a user interface, and independent polymer table-row element, and server and client side Dart code.

The main.html file is first loaded and provides the user with a drop down select html element that gives an option of the various test files. The client.dart script is run with the main.html code and listens for a change in the select element, signifying that the user has selected a new DICOM file to read. On a change in the select element the client.dart code sends a POST request to the server with the information from the selector identifying which file was selected.

Figure 3: Overview of Data Pipeline



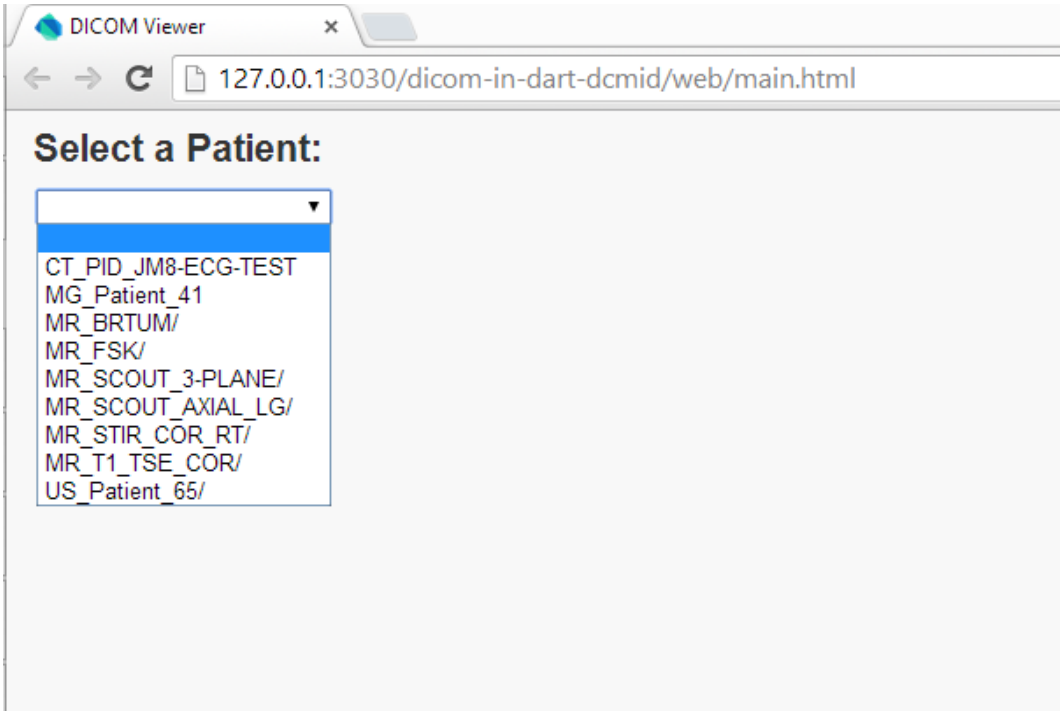
When the server starts running, which should happen before the client accesses the web page itself, it populates a list of DICOM files by recursively searching the data directory looking for files that have the .msdicom extension. This makes it easy for an administrator to simply add files to that directory and the server handles the implementation automatically. Once the POST request has been completed client.dart creates a GET request for the actual bytes stored in the file that was selected through the previous POST request. To optimize the code further it is desirable to consolidate these requests into one GET request, but Dart does not seem to handle the pushing and pulling of data well at this point in time.

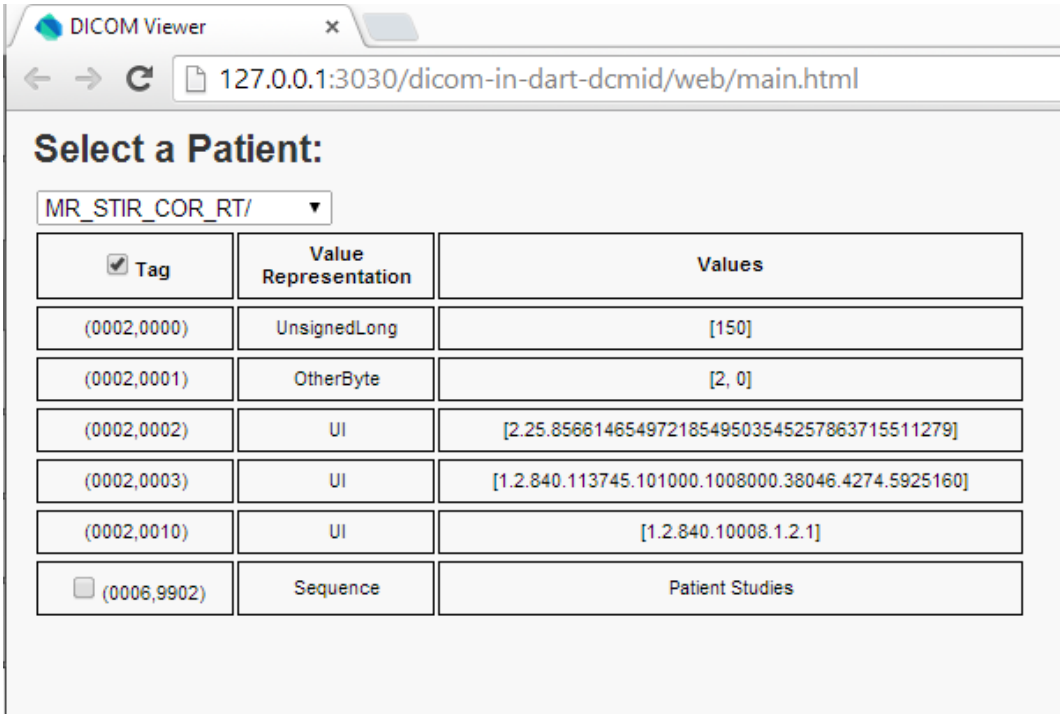
A ByteBuf and subsequent Dataset is created in client.dart and this Dataset is then read on the client side. Once the Dataset is read the contents of the Dataset are recursively added to the main.html as table-row polymer elements at the appropriate nesting levels. This is then dynamically displayed on the end point device so that the user can interact with the final, now readable, DICOM file.

3.4 The Interface

The user interface was designed to allow a user-friendly display off the DICOM metadata. It displays the attributes, each of which has a tag, a value representation, and a value. Upon loading the application, the user chooses a patient from the drop down list. The selection then shows a tree-table structure that displays the metadata. Clicking on a checkbox, which is present next to

sequence attributes, will result in displaying all of the sequences children at the next level of the tree.





4 Results and Discussion

Using the Dart, code was developed that is able to parse and display binary DICOM data in a web browser. This is believed to be the first time this has been accomplished. An original, stand-alone HTML5 DOM element using Polymer in order to display the metadata in a collapsible tree-table view was also developed. This code will also be made open-source and can be used independently of the main web application. Figure 3 shows the average time it takes to parse each test file versus the file size. During the time working on this project access to 14 test files was obtained, with sensitive patient information removed. The de-identified test data ranged from 3.9 KB to 3.99 MB. For the cases smaller than 0.1 MB, the program was able to parse the metadata in less than 0.5 seconds. The program parsed the metadata of the largest file in approximately 2.5 seconds.

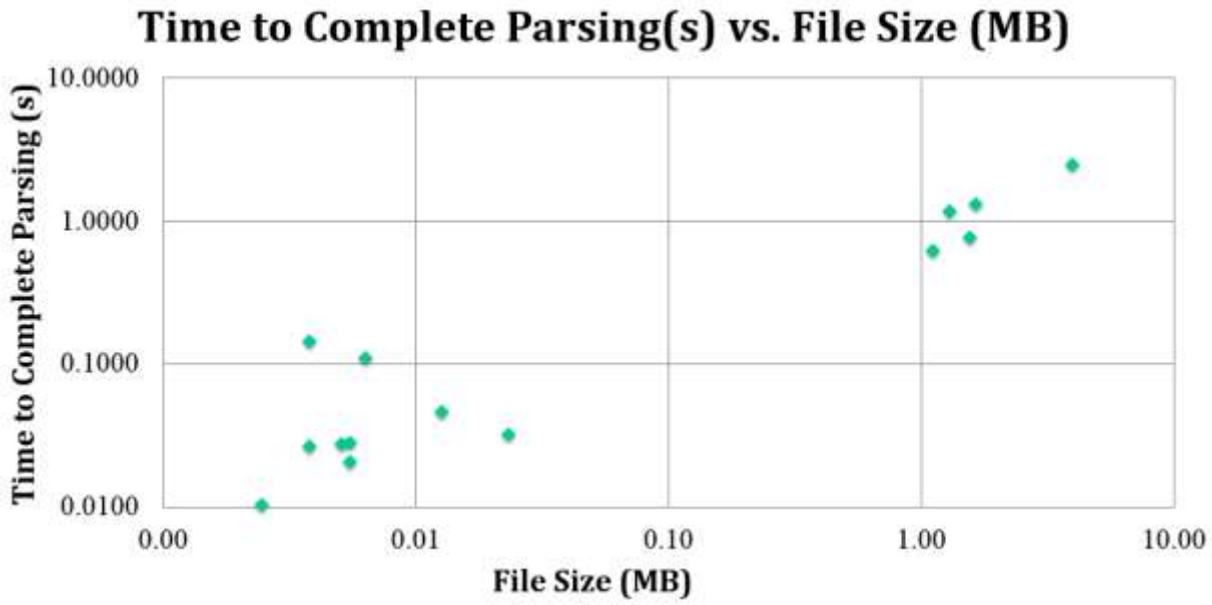


Figure 4: Time to complete parsing vs. file size for test DICOM files.

Both axes are plotted using a logarithmic scale. Metadata files smaller than 0.1 MB are able to be parsed in less than 0.5 seconds. The largest file (3.99 MB) is parsed in approximately 2.5 seconds.

The system can parse and display all DICOM attribute formats. The software makes it very easy to retrieve and display the data that is stored on the server side to the client where the parsing occurs very quickly. Currently a small problem that is being worked on is the speed of the

remaining pipeline, before and after the data is parsed. During testing these additional time costs such as the time of server-client communication, data transfer, and recursive display can add a significant wait time for the user to view their files. Once Dart's server and client best practices become more flushed out and understood, these issues should be solved.

5 Conclusion

In this project, a novel application has been proposed for the viewing of binary DICOM medical imaging studies. This application has demonstrated that the new programming language Dart can be used to parse the data within the goal time of 3 seconds. The application can parse all DICOM formats and the server and client code make it easy for both a system administrator as well as the end point user to interact with DICOM in an intuitive manner.

The software was written under strict time constraints and has much more room for optimization in both parsing as well as server-client interaction. There was a lot of work done to give this application many of the parts that it should have that were not as relevant to the main part of the project in an attempt to make future development much easier for Dr. Philbin and future programmers. A lot of code exists that has yet to be implemented in the main methods that have been written as standalone classes. Once development advances further, these will prove invaluable and contribute significantly to the end project.

6 Management Summary

6.1 Division of Tasks

Damish and Danielle originally met with mentor, Dr. Philbin, once a week, and progressed to twice weekly meetings to discuss issues concerning library structure. Damish and Danielle met with each other three times weekly to work on and test code.

Originally, both Damish and Danielle were to work on all parts of the project together, first by working on the parsers and assigning classes for each person to complete. However the team

split up the larger goals (of finishing the parsers and creating the user interface) and worked separately while checking the other person's code. Damish continued to work on the parsers, with constant revision suggested by our mentor, while Danielle worked on the user interface. After the parsers were completed, Damish worked on setting up the server-client relationship while Danielle wrote the API.

6.2 What was Accomplished

The original plan was to build a DICOM viewer/editor. However, work to edit metadata and display bulkdata in the web browser was delayed as a result of underestimating the amount of time required to build the moving parts in this project using Dart. In particular, our team was not expecting to have to build an entire ByteBuf base class to handle the data directly. This addition, and making sure it was written properly, added a significant amount of time to our final project timeline. We favored creating a web viewer application with the intention of iteratively building upon its structure in the future. Because it is still in its early stages, the project will not yet be made public, but is still being developed to become open-source. Looking at the source code together with the API we can see that the main classes used are as follows:

- ByteBuf – Base class that handles our DICOM files and contains general binary parsers.
- Dataset – Contains DICOM specific parsers and reading methods for datasets and attributes.
- Table-row – Polymer element that is used to store our attributes and display them in html.
- Server – Contains all the code necessary for server operation. Automatically finds DICOM metadata in a given directory.
- Client + Main – Create initial display and parse through the DICOM data requested from the server. Also handles displaying the newly parsed data.
- A large number of helper classes that:
    - Add utility features to our ByteBuf and Dataset classes.
    - Contain DICOM specific classes to represent specific types of data.
    - Classes such as Information Entities that have yet to be implemented.
- Unittests were written to verify functional parsers.

6.3 Project Timeline

	Feb		Mar				Apr				May	
	20	27	6	13	20	27	3	10	17	24	1	9
Project Proposal	Done!											
Read input (parse)							Done!					
Validate output								Done!				
Display metadata in browser											Done!	
Polymer UI Element											Done!	
Display images												Future work.
Clean Code and Finish API												Done!
Final Presentation												Done!

6.4 Deliverables

Expected deliverables (April 28)

- ✓ Display a working list of studies of n patients
- ✓ Display patient as collapse/expand tree for study information model
- ✓ Develop a stand-alone collapsible tree-table Polymer UI Element
- ✓ Clean code and finish API
- ✓ Create a user domain

Maximum deliverables (May 1) → Future Work

- – Display images
- – Edit metadata
- – Encrypt and decrypt studies using AES (GCM) using an encryption framework created at Hopkins Security Institute

Minimum deliverables (April 5)

- ✓ Read and display DICOM in a browser and then write it
- ✓ Build a test program that compares input and output to validate correctness and display DICOM in a browser and then write it
- ✓ Create unit tests for each class

7 Future Work

A future goal of this project would be to increase the speed of the server-client interaction. Since Dart is still fairly young and the team ran into some time issues, the server-client interaction through Dart code is still not fully understood. In particular we could greatly speed up this part of the code through consolidation of our GET and POST requests to the server into a single GET request. This will likely be a main goal of future development. Another place that the code could be optimized is in using the developed information entity structure. Currently the data is being stored using only the Dataset class and is recursively storing a new list of Datasets whenever a Sequence is encountered. This recursive structure is then being read and is displayed through more recursive code. Storing it in objects that reflect the information entity structure would help consolidate some of that time complexity. Future work will also include the original maximum deliverables of being able to display the bulkdata, edit the metadata, adding overlay information to the bulkdata, and being able to encrypt and decrypt the studies.

References

[1] DICOM: DICOM standard. DICOM (Digital Imaging and Communications in Medicine), Part 3: Information Object Definitions, Rosslyn, VA 2011. http://medical.nema.org/standard.html.

[2] Google. Dart VM and dart2js Performance. https://www.dartlang.org/performance/
Polymer Library. http://www.polymer-project.org/

Mahmoud Ismail, Yu Ning, and James Philbin, Separation of metadata and pixel data to speed DICOM tag morphing. *SPIE Medical Imaging 2014: PACS and Imaging Informatics: Next Generation and Innovations*, Forthcoming.

Mahmoud Ismail, Yu Ning, James Philbin. Transmission of DICOM Studies using Multi-Series DICOM Objects. *Proceedings SPIE 8674, Medical Imaging 2013: Advanced PACS-based Imaging Informatics and Therapeutic Applications*. April 8, 2013.

Appendix

Bitbucket: https://bitbucket.org/ohtmiami/dicom-in-dart-dcmid

API: Please refer to Excel sheet DCMiD API